JPL PUBLICATION 85-23

(NASA-CR-175872) A SOFTWARE SIMULATION STUDY OF A (255,223) REED-SOLOMON ENCODER-DECODER (Jet Propulsion Lab.) 43 p HC A03/MF A01 CSCL 17B

N85-28192

Unclas
G3/32 21439

A Software Simulation Study of a (255,223) Reed-Solomon Encoder/Decoder

Fabrizio Pollara

April 15, 1985

NASA
National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

JPL PUBLICATION 85-23

A Software Simulation Study of a (255,223) Reed-Solomon Encoder/Decoder

Fabrizio Pollara

April 15, 1985

National Aeronautics and
Space Administration

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

The research described in this publication was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

FIGURES

TABLES

ABSTRACT

A set of software programs which simulates a (255,223) Reed-Solomon encoder/decoder pair is described. The transform decoder algorithm uses a modified Euclid algorithm, and closely follows the pipeline architecture proposed for the hardware decoder. Uncorrectable error patterns are detected by a simple test, and the inverse transform is computed by a finite field FFT.

Numerical examples of the decoder operation are given for some test codewords, with and without errors. The use of the software package is briefly described.

1. INTRODUCTION

A (255,223) Reed-Solomon (RS) code has been adopted as the standard outer code for concatenated coding systems by NASA and by the European Space Agency (ESA) [1]. This particular RS code is defined in $GF(2^8)$ by the following parameters:

$N = 255$ = number of 8-bit symbols in a codeword (block)

$K = 223$ = number of information symbols in a block

$T = N-K$ = number of parity symbols.

Such a code is capable of correcting up to $T/2 = 16$ symbol errors in a block. The generator polynomial $g(x)$ of the code is given by,

$$g(x) = \prod_{i=M}^{M+T+1} (x - \alpha^{Gi}) = \sum_{j=0}^{T} g_j x^j \tag{1}$$

where $M = 112$, $G = 11$, and α is a root of the primitive polynomial over $GF(2)$

$$x^8 + x^7 + x^2 + x + 1$$

Every element of $GF(2^8)$ can be represented as a polynomial in α over $GF(2)$ of degree less than 8, as shown in Table 1.

The constant M is chosen so that the polynomial has symmetrical coefficients, i.e.,

$$g_j = g_{T-j},\ j=0,1,\ldots,T$$

It is shown in [2] that this is true if $M = 2^{8-1} - (T/2) = 112$.

The constant $G = 11$ is chosen to minimize the bit-serial implementation complexity of the encoder [3]. The polynomial coefficients are shown in Table 2.

Table 1. Decimal Representation of Elements of GF(2^8)

$z = \alpha^x; \ x = \alpha^y$

x	y	z	x	y	z	x	y	z	x	y	z
0	*	1	1	0	2	2	1	4	3	99	8
4	2	16	5	198	32	6	100	64	7	106	128
8	3	135	9	205	137	10	199	149	11	188	173
12	101	221	13	126	61	14	107	122	15	42	244
16	4	111	17	141	222	18	206	59	19	78	118
20	200	236	21	212	95	22	189	190	23	225	251
24	102	113	25	221	226	26	127	67	27	49	134
28	108	139	29	32	145	30	43	165	31	243	205
32	5	29	33	87	58	34	142	116	35	232	232
36	207	87	37	172	174	38	79	219	39	131	49
40	201	98	41	217	196	42	213	15	43	65	30
44	190	60	45	148	120	46	226	240	47	180	103
48	103	206	49	39	27	50	222	54	51	240	108
52	128	216	53	177	55	54	50	110	55	53	220
56	109	63	57	69	126	58	33	252	59	18	127
60	44	254	61	13	123	62	244	246	63	56	107
64	6	214	65	155	43	66	88	86	67	26	172
68	143	223	69	121	57	70	233	114	71	112	228
72	208	79	73	194	158	74	173	187	75	168	241
76	80	101	77	117	202	78	132	19	79	72	38
80	202	76	81	252	152	82	218	183	83	138	233
84	214	85	85	84	170	86	66	211	87	36	33
88	191	66	89	152	132	90	149	143	91	249	153
92	227	181	93	94	237	94	181	93	95	21	186
96	104	243	97	97	97	98	40	194	99	186	3
100	223	6	101	76	12	102	241	24	103	47	48
104	129	96	105	230	192	106	178	7	107	63	14
108	51	28	109	238	56	110	54	112	111	16	224
112	110	71	113	24	142	114	70	155	115	166	177
116	34	229	117	136	77	118	19	154	119	247	179
120	45	225	121	184	69	122	14	138	123	61	147
124	245	161	125	164	197	126	57	13	127	59	26
128	7	52	129	158	104	130	156	208	131	157	39
132	89	78	133	159	156	134	27	191	135	8	249
136	144	117	137	9	234	138	122	83	139	28	166
140	234	203	141	160	17	142	113	34	143	90	68
144	209	136	145	29	151	146	195	169	147	123	213
148	174	45	149	10	90	150	169	180	151	145	239
152	81	39	153	91	178	154	118	227	155	114	65
156	133	130	157	161	131	158	73	129	159	235	133
160	203	141	161	124	157	162	253	189	163	196	253
164	219	125	165	30	250	166	139	115	167	210	230
168	215	75	169	146	150	170	85	171	171	170	209
172	67	37	173	11	74	174	37	148	175	175	175
176	192	217	177	115	53	178	153	106	179	119	212
180	150	47	181	92	94	182	250	188	183	82	255
184	228	121	185	236	242	186	95	99	187	74	198
188	182	11	189	162	22	190	22	44	191	134	88
192	105	176	193	197	231	194	98	73	195	254	146
196	41	163	197	125	193	198	187	5	199	204	10
200	224	20	201	211	40	202	77	80	203	140	160
204	242	199	205	31	9	206	48	18	207	220	36
208	130	72	209	171	144	210	231	167	211	86	201
212	179	21	213	147	42	214	64	84	215	216	168
216	52	215	217	176	41	218	239	82	219	38	164
220	55	207	221	12	25	222	17	50	223	68	100
224	111	200	225	120	23	226	25	46	227	154	92
228	71	184	229	116	247	230	167	105	231	193	210
232	35	35	233	83	70	234	137	140	235	251	159
236	20	185	237	93	245	238	248	109	239	151	218
240	46	51	241	75	102	242	185	204	243	96	31
244	15	62	245	237	124	246	62	248	247	229	119
248	246	238	249	135	91	250	165	182	251	23	235
252	58	81	253	163	162	254	60	195	255	183	99

Table 2. Coefficients of Generator Polynomial

g_0	=	g_{32}	=	α^{0}
g_1	=	g_{31}	=	α^{249}
g_2	=	g_{30}	=	α^{59}
g_3	=	g_{29}	=	α^{66}
g_4	=	g_{28}	=	α^{4}
g_5	=	g_{27}	=	α^{43}
g_6	=	g_{26}	=	α^{126}
g_7	=	g_{25}	=	α^{251}
g_8	=	g_{24}	=	α^{97}
g_9	=	g_{23}	=	α^{30}
g_{10}	=	g_{22}	=	α^{3}
g_{11}	=	g_{21}	=	α^{213}
g_{12}	=	g_{20}	=	α^{50}
g_{13}	=	g_{19}	=	α^{66}
g_{14}	=	g_{18}	=	α^{170}
g_{15}	=	g_{17}	=	α^{5}
g_{16}	=	α^{24}		

The algorithm used is a transform decoder as described in [4], which is based on a modified Euclid algorithm to compute the error locator polynomial. Therefore, this simulation can be used to verify the performance of the proposed pipeline hardware decoder.

The only modifications consist in adapting the algorithm to symmetric generator polynomials, using a finite field FFT (Fast Fourier Transform) to compute the error pattern, and testing for uncorrectable error patterns.

2. SIMULATION SET-UP

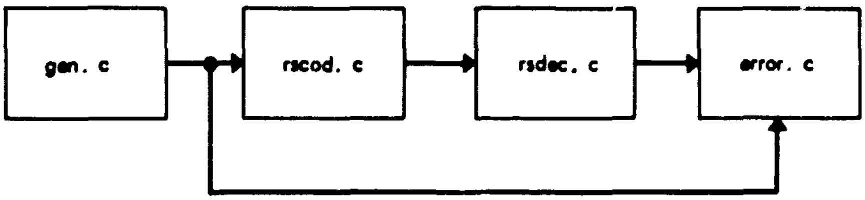
The set of software subroutines includes a random generator (gen.c) of sequences taken from {0,1}, a RS encoder (rscod.c), a RS decoder (rsdec.c), and a block (error.c) which computes bit and symbol error probabilities. These subroutines are called by a main program named "universe.c".

All programs are written in C-language on a VAX 750 computer. Figure 1(a) shows the block diagram of the simulation set-up. Channel errors are artificially introduced at the input of the RS decoder. If desired, the set-up may be modified to that of Fig. 1(b), where errors are produced by adding a sequence of random variables (for example Gaussian, if the subroutine "gauss.c" is used) to the encoded stream. Error bursts may be added with a separate subroutine, or by a concatenated, convolutional code and Viterbi decoding.

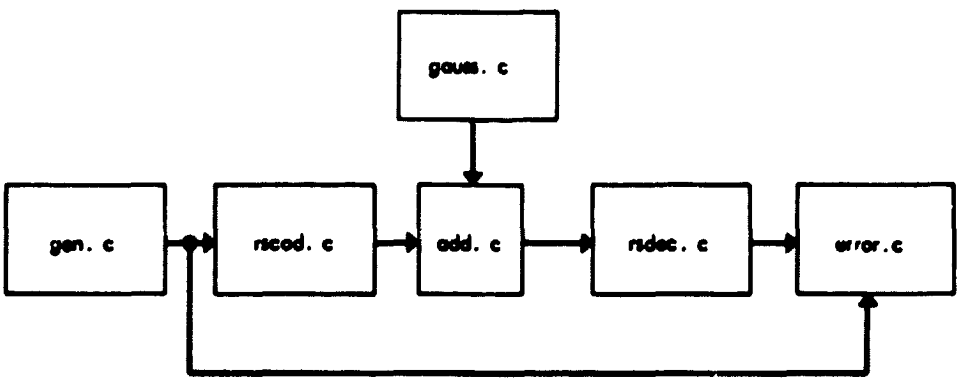
The modularity of the program allows the simulation of concatenated coding schemes to be described in a separate report.

3. RS ENCODER

Since we are considering a systematic RS code, the encoder will first output the K information symbols a_j. The T parity symbols are the coefficients b_i of the remainder polynomial $b(x) = b_0 + b_1 x + \dots + b_{T-1} x^{T-1}$, resulting from dividing the message polynomial $x^T a(x)$ by the generator polynomial $g(x)$, where $a(x) = a_0 + a_1 x + \dots + a_{K-1}x^{K-1}$.



(a) Fixed Error Pattern



(b) Random Error Pattern

Fig. 1. Simulation Block Diagram

This polynomial division can be easily implemented by a shift register divider, as shown in the logic diagram of Fig. 2, for the (255,223) RS code. Additions are to be interpreted modulo-two (exclusive-OR), multiplications in the field are performed by table look-up, where the table is automatically constructed during the first execution. The subroutine listing is shown in Appendix B.

The algorithm proceeds as follows:

(0) Initialize $b_i = 0$, $i=0,\ldots,T$

(A) During the first 223 iterations ($0 \le j \le 222$):

(1) get information symbol a_j

(2) $v = a_j + b_{T-1}$

(3) output $z = a_j$

(B) During last 32 iterations ($224 \le j \le 255$):

(1) $v = 0$

(2) output $z = b_{T-1}$

(C) For all j's:

(1) $b_i = b_{i-1} + (g_i * v)$, $i=T-1,T-2,\ldots,1$

(2) $b_0 = v$

The encoder may be tested by forcing the generator to produce some given pattern whose corresponding codeword is known, and printing the output

4. RS DECODER

4.1 DECODER ALGORITHM

The decoder performs the following basic operations:

- get received codeword
- compute syndrome
- obtain the error-locator polynomial by using the modified Euclid algorithm
- compute the remaining elements of the error sequence transform
- compute the inverse transform yielding the estimated error pattern.

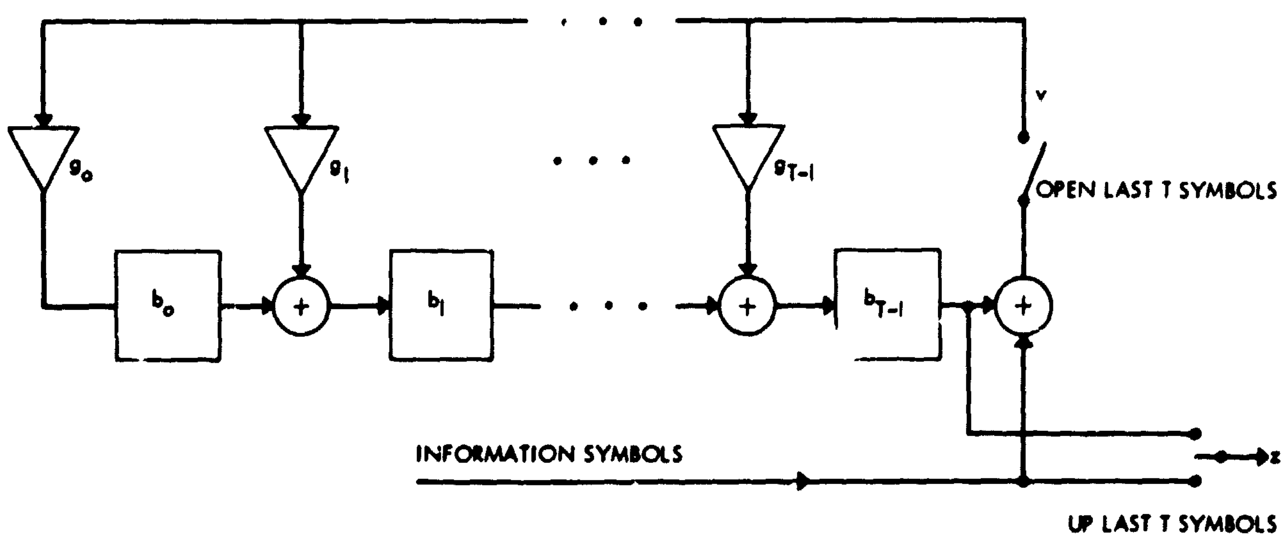


Fig. 2. RS Encoder

Consider the generator polynomial in (1), and define:

$\underline{U} = [u_0, u_1, \ldots, u_{N-1}]$ = received codeword

where $\begin{cases} [u_0, u_1, \ldots, u_{N-T-1}] & = \text{information symbols} \\ [u_{N-T}, \ldots, u_{N-1}] & = \text{parity} \end{cases}$

$\underline{S} = [s_0, s_1, \ldots, s_{T-1}]$ = syndrome

$\underline{R} = [r_0, r_1, \ldots, r_T]$

$\underline{\lambda} = [\lambda_0, \lambda_1, \ldots, \lambda_{T-1}]$ (contains error-locator polynomial at last iteration)

$\underline{\mu} = [\mu_0, \mu_1, \ldots, \mu_{T-1}]$

$\underline{E} = [E_0, E_1, \ldots, E_N]$ = error pattern transform

$\underline{e} = [e_0, e_1, \ldots, e_N]$ = error pattern

$d(\underline{S}) = \{ j : s_j \neq 0 \text{ and } s_i = 0 ,\ j<i<T \}$

and similarly for $d(\underline{R})$ and $d(\underline{\lambda})$.

Then the decoder algorithm can be described as follows:

(1) get received codeword $\underline{U}$

(2) compute the syndrome, (see Appendix A)

$\underline{S} = \underline{0}$

$s_j = u_{N-1-j} + \alpha^{G(j+M)} s_j$; $j=0,\ldots,T-1$; $i=0,\ldots,N-1$

(3) if $d(\underline{S})=0$ go to (11)

(4) initialize,

$E_{j+1} = s_{T-1-j}$; $j=0,\ldots,T-1$

$\underline{R} = \underline{0}$

$r_T = 1$

$\underline{\mu} = \underline{0}$

$\mu_0 = 1$

$\underline{\lambda} = \underline{0}$

$i = 1$

(5) **while ($i<T$) do:**

$L = d(\underline{R}) - d(\underline{S})$

if $d(\underline{R})<T/2$ go to (6)

else if $d(\underline{S})<T/2$, $\underline{\lambda} = \underline{\mu}$, go to (6)

else do EUCLID (see section 4.2)

$i \leftarrow i+1$

(6) compute normalized error-locator polynomial

$$B = \alpha^{-\lambda_{d(\underline{\lambda})}}$$

$$\lambda_j = B\lambda_j \;;\; j=0,\ldots,d(\underline{\lambda})$$

(7) compute remaining elements of error transform

$$\left.\begin{array}{l} E_{j+1} = 0 \\ E_{j+1} = E_{j+1} + \lambda_{d(\lambda)-1-i}\, E_{j-1} \;;\; i=0,\ldots,d(\underline{\lambda})-1 \\ E_0 = E_n \end{array}\right\} j=T,\ldots,N+T-1$$

(8) test for uncorrectable error patterns,

if $E_j \neq E_{j+N}$ for some $j=1,\ldots,T$, go to (11)

(9) compute inverse transform $\underline{e}$ (see section 4.3)

(10) compute corrected sequence,

$\underline{U} = \underline{U} + \underline{e}$

(11) output $\underline{U}$

A complete listing of the decoder subroutine is shown in Appendix C.

The test in step (8) is explained in [6]. It may also be observed that this RS code is effective in terms of undetected errors, since [7, 8], for independent symbol errors, the probability of undetected error P_u is bounded by:

$$P_u < (N+1)^{-T} \sum_{i=0}^{T/2} \binom{N}{i} N^i < \frac{1}{(T/2)!}$$

For the (255,223) code, $P_u < 4.8\ 10^{-14}$. But, for the (15,9) code considered in [4], $P_u < 0.093$.

4.2 EUCLID ALGORITHM

This is a modified version of Euclid's algorithm for polynomials [5], which does not need the computation of inverse field elements. It operates on two polynomials,

$$A(x) = x^T \quad \text{and} \quad S(x) = \sum_{K=1}^{T} a_K\, x^{T-K}$$

and finds the i^{th} remainder $R_i(x)$ of degree less than T/2, satisfying:

$$\gamma_i(x)\, A(x) + \lambda_i(x)\, S(x) = R_i(x)$$

At the end, $\lambda_j(x)$ is the desired (unnormalized) locator polynomial. The algorithm is implemented as follows:

$$\text{if } d(\underline{R}) < d(\underline{S}) \quad \left\{ \begin{array}{l} \underline{R} \longleftrightarrow \underline{S} \\ \underline{\lambda} \longleftrightarrow \underline{\mu} \\ d(\underline{R}) \longleftrightarrow d(\underline{S}) \end{array} \right.$$

$$\text{if } s_{d(\underline{S})} = 0 \quad \left\{ \begin{array}{l} d(\underline{S}) \longleftarrow d(\underline{S}) - 1 \\ \\ \text{if } d(\underline{S}) < T/2,\ \underline{\lambda} = \underline{\mu},\ \text{return} \end{array} \right.$$

$$\text{else} \quad \left\{ \begin{array}{l} a = r_{d(\underline{R})}\ ;\ b = s_{d(\underline{S})} \\ d(\underline{R}) \longleftarrow d(\underline{R}) - 1 \\ \hat{\underline{S}} = D_{|L|}\ (\underline{S}) \\ \underline{R} = b\,\underline{R} + a\,\hat{\underline{S}} \\ \hat{\underline{\mu}} = D_{|L|}\ (\underline{\mu}) \\ \underline{\lambda} = b\,\underline{\lambda} + a\,\hat{\underline{\mu}} \\ \text{if } d(\underline{R}) < T/2\ ,\ \text{return} \end{array} \right.$$

where $D_{|L|}(\underline{x})$ shifts right the components of a vector $\underline{x}$ by $|L|$ positions, and fills with zeros.

4.3 INVERSE FFT

A direct computation of the inverse transform,

$$e_j = \sum_{i=0}^{N-1} \alpha^{Gij}\, E_{N+1+i+M};\ i=0,\ldots,N-1;\ j=0,\ldots,N-1$$

requires $N^2 = 65025$ multiplications. The number of multiplications may be reduced by organizing the N-point one-dimensional array $\underline{E}$ into a two-dimensional $n_1 \times n_2$ array, where $n_1 n_2 = N$, and n_1 and n_2 are

relatively prime. This algorithm (Good-Thomas FFT [6]) is based on the Chinese remainder theorem.

Let $b = (a)_N$ denote the remainder of a modulo N, and define

$$i_1 = (i)_{n_1},\ i_2 = (i)_{n_2},\ j_1 = (\tilde{N}j)_{n_1},\ \text{and } j = (\tilde{N}j)_{n_2}$$

Then,

$$i = (\tilde{N}(n_2 i_1 + n_1 i_2))_N \quad \text{and} \quad j = (n_2 j_1 + n_1 j_2)_N$$

where, $(\tilde{N}(n_1 + n_2)) = 1 \Longrightarrow \tilde{N} = 8.$

Now the inverse transform may be written in the following two steps

$$D_{i_1,j_2} = \sum_{i_2=0}^{n_2-1} E_{N+1-M+i}\,\alpha^{Gn_1 i_2 j_2} \qquad 0 < i_1 < n_1\ ,\ 0 < j_2 < n_2$$

$$e_j = \sum_{i_1=0}^{n_1-1} D_{i_1,j_2}\,\alpha^{Gn_2 i_1 j_1} \qquad 0 < j_1 < n_1\ ,\ 0 < j_2 < n_2$$

For $N = 255 = 17 \cdot 15 = n_1 n_2$, the number of multiplications is reduced from N^2 to $N(n_1 + n_2)$. A further reduction may be obtained, if desired, by factoring N as $N = 17 \cdot 5 \cdot 3$.

5. USER GUIDE AND EXAMPLES

This software package may be run on any computer having a C-language compiler. The source code for the full set of subroutines is available by contacting the author. Subroutines required are:

(block management routines in object code form)

sequencer.o
block.o
fifo.o

(include files)

star.h

dstar.h

type.h

alloc.h

para.h

param.h

dfifo.h

(simulation blocks)

gen.c

rscod.c

rsdec.c

add.c

gauss.c

error.c

universe.c

Subroutines are also available to simulate the (15,9) RS code considered in [4]. If the UNIX operating system is used, it is advisable to create a "makefile" to maintain (compile and link) the set of subroutines. In any case the subroutines must be compiled and linked to produce an executable image file.

In order to run the simulation it is not necessary to provide any external parameter or data file, since the information symbols are generated internally. If specific information sequences are of interest, the subroutine "gen.c" can be easily modified for this purpose. Non-real-time decoding of actual data could be accomplished by modifying "rsdec.c" so that it will read the data from a disk file in segments of a given number of blocks.

The output contains the number of channel symbol errors, and the number of corrected symbol and bit errors. If the number of channel errors is greater than T/2, a warning message is printed. If real data needs to be

decoded, the decoded symbols can be displayed by adding a print statement in "rsdec.c". All output is normally displayed on the standard output (CRT), but it can be redirected to a disk file through the operating system. As an example, under UNIX, we could type:

```
sim > outfile
```

where "sim" is the executable program and the file "outfile" will contain the output.

By including the print statements provided in the subroutine rsdec.c, it is possible to display all the intermediate steps of the decoder. Such an example of output is shown in Appendix D for a given codeword and the error pattern $e_7 = \alpha^{202}$, $e_{120} = \alpha^0$, $e_i = 0$, $i \neq 7, 120$. Elements in GF(256) are represented in decimal base.

If no errors were present, the output would show that $\underline{S} = \underline{0}$.

A randomly chosen codeword is shown in Appendix E.

6. REFERENCES

[1] "Recommendation for Space Data System Standards: Telemetry Channel Coding," (Blue book), Consultative Committee for Space Data Systems, May 1984.

[2] Berlekamp, E.R., Algebraic Coding Theory, McGraw-Hill, N.Y., 1968.

[3] Berlekamp, E.R., "Bit-Serial Reed-Solomon Encoders," IEEE Transactions on Information Theory, Nov. 1982, p. 869.

[4] Shao, H.M., et. al., "A Systolic VLSI Design of a Pipeline Reed-Solomon Decoder," TDA Progress Report 42-76, JPL, Feb. 1984, p. 99.

[5] Brent, R.P. and H.T. Kung, "Systolic VLSI Arrays for Polynomial GCD Computations," Computer Science Dept. Report, Carnegie-Mellon Univ., Pittsburg, PA, 1982.

[6] Blahut, R.E., Theory and Practice of Error Control Codes, Addison-Wesley, London, 1983.

[7] Swanson, L. and R. McEliece, "Reed-Solomon Codes for Error Detection," 1985 International Symp. on Inform. Theory, in press.

[8] Kasami, T. and S. Lin, "On the Probability of Undetected Error for Maximum Distance Separable Codes," IEEE Trans. on Comm., Vol. COM-32, Sept. 84, p. 998.

APPENDIX A

Number of Multiplications Required to Compute the Syndrome of a (255,223) RS Code with a Symmetric Generator Polynomial

Consider the generator polynomial in (1), then the syndrome is defined as,

$$s_j = \sum_{i=0}^{N-1} u_i \alpha^{Gi(j+M)}, \quad j=0,\ldots,T$$

Define the NxN matrix J,

$$J = \begin{bmatrix} 0 & & 1 \\ & \cdot^{\cdot^{\cdot}} & \\ 1 & & 0 \end{bmatrix}$$

such that,

$$\tilde{\underline{u}} = [\tilde{u}_0, \ . \ . \ . \ , \tilde{u}_{N-1}] = [u_{N-1}, \ . \ . \ . \ , u_0] = \underline{u}\ J$$

Then we can consider a new syndrome $\tilde{\underline{S}}$

$$\tilde{s}_m = \sum_{i=0}^{N-1} \tilde{u}_i \ \alpha^{Gi(m+M)}, \quad m=0,\ldots,T-1$$

$$= \sum_{i=0}^{N-1} u_{N-i-1} \alpha^{Gi(m+M)}, \quad m=0,\ldots,T-1$$

Let $k = N - i - 1$

$$\tilde{s}_m = \sum_{k=0}^{N-1} u_k \alpha^{G(N-1-k)(m+M)} = \alpha^{-G(m+M)} \sum_{k=0}^{N-1} u_k \alpha^{-G(m+M)}$$

But,

$$\alpha^{-G(m+M)} = \alpha^{G(h+M)}, \text{ where } m = T - 1 - h, \ h=0,\ldots,T-1$$

$$\longrightarrow \qquad \tilde{s}_{T-1-h} = \alpha^{G(h+M)} \sum_{k=0}^{N-1} u_k \alpha^{GK(h+M)}, \quad h=0,\ldots,T-1$$

And finally, $\tilde{s}_{T-1-h} = \alpha^{G(h+M)} s_h$, $h=0,\ldots,T-1$

$$\underline{\tilde{S}}J = \underline{S}\Gamma, \text{ where } \Gamma = \begin{bmatrix} \alpha^{GM} & & & 0 \\ & \alpha^{G(M+1)} & & \\ & & \ddots & \\ 0 & & & \alpha^{G(M+T-1)} \end{bmatrix}$$

$$\underline{S} = \underline{u}A,$$

where $A = [a_{ij}]$, $a_{ij} = a^{Gi(j+M)}$, $i=0,\ldots,N-1$, $j=0,\ldots,T-1$

$$\underline{\tilde{S}} = \underline{u}JA$$

$$\underline{S}(J+\Gamma) = \underline{u}AJ + \underline{u}JAJ = \underline{u}(AJ + JAJ) \triangleq \underline{d}$$

Let

$$B = AJ+JAJ = \left[\, \underline{b}_0 \cdots \underline{b}_j \cdots \underline{b}_{T-1} \right] \triangleq \left[\cdots \left| \begin{matrix} \underline{b}^*_j \\ b^*_j \\ J\underline{b}^*_j \end{matrix} \right| \cdots \right]$$

be a partition of B into the column vectors $\underline{b}_j$, and

$$\underline{u} = [\underline{u}_1, u_c, \underline{u}_u]$$

Then $\quad d_j = \underline{u}_1 b^*_j + u_c b^* + \underline{u}_u Jb^*_j = (\underline{u}_1 + \underline{u}_u J)\underline{b}^*_j + u_c b^*$

The computation of d_j requires only 254/2 + 1 = 128 multiplications (instead of 255).

$$\underline{S} = \underline{d}\,(J+\Gamma)^{-1}$$

Note that $(J+\Gamma)^{-1}$ has the form:

$$(J+\Gamma)^{-1} = \begin{bmatrix} \beta_0 & & & 0 & & \epsilon_{T-1} \\ & \beta_1 & & & \cdot & \\ & & \ddots & \cdot & & \\ 0 & & \vdots & & & 0 \\ & \epsilon_1 & & & \ddots & \\ \epsilon_0 & & & 0 & & \beta_{T-1} \end{bmatrix}$$

Therefore,

$$S_j = \beta_j d_j + \epsilon_j d_{T-1-j}, \quad j=0,\ldots,T-1$$

So that each S_j can be computed with 128 + 2 = 130 multiplications.

APPENDIX B

RS Encoder Subroutine

```
/*****    Reed-Solomon Encoder   ( CCSDS Doc. #1 , Sept 1983)  ***/

#include <stdio.h>
#include "../type.h"
#include "../star.h"
#define TT      32
#define N       255
#define K       223
#define CNT     pstate->cnt
#define V       pstate->vv
#define G       pstate->g
#define B       pstate->b
#define H       pstate->h
#define F       pstate->f
typedef struct {
        int non;
}
PARAM, *PARAMPTR;
typedef struct {
        unsigned char b[TT],g[TT+1],cnt,h[N],f[N+1],vv;
}
STATE, *STATEPTR;
rscod (pparam,size,pstate,pstar)
PARAMPTR pparam;
STATEPTR pstate;
STARPTR pstar;
int size;
{
        SAMPLE x;
        int i,j;

        if (pstate == NULL) {
                pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
                if (no_input_fifos( ) !=1 || no_output_fifos( ) !=2)
                        return(3);

/*** H[ ] and F[ ] compute the power and log in GF(256)  ********/

                H[0]=1;
                for(i=0;i<8;i++) H[i+1]=2*H[i];
                for(i=8;i<N;i++) H[i]=H[i-1]^H[i-6]^H[i-7]^H[i-8];

                for(j=1;j<N+1;j++)  {
                        for(i=0;i<N;i++)  {
                                if(H[i]==j) F[j]=i;

                        }
                }

                CNT=0;
                V=0;
                F[0]=0;
```

```
/***  G[ ] are the coefficients of the generating polynomial ****/
                G[0]=H[0];
                G[1]=H[249];
                G[2]=H[59];
                G[3]=H[66];
                G[4]=H[4];
                G[5]=H[43];
                G[6]=H[126];
                G[7]=H[251];
                G[8]=H[97];
                G[9]=H[30];
                G[10]=H[3];
                G[11]=H[213];
                G[12]=H[50];
                G[13]=H[66];
                G[14]=H[170];
                G[15]=H[5];
                G[16]=H[24];
                for(i=0;i<TT/2;i++) G[TT-i]=G[i];
        }

 if(length_output_fifo(0) != length_output_fifo(1)) return(7);
 if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
 if (length_output_fifo(1)==maxlength_output_fifo(1)) return(0);

        while(length_input_fifo(0)  >0  || CNT>=K )
        {
 if(length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
 if(length_output_fifo(1)==maxlength_output_fifo(1)) return(0);
        if(CNT==0)    for(i=0;i<TT;i++) B[i]=0;
/*******************************************************************/
                if(CNT<K)                   /** information bits **/
                {
                        get(0,&x) ;
                        V=((int)x)^B[TT-1];
                }
                else                        /** parity bits **/
                {
                        V=0;
                        x=(SAMPLE)B[TT-1];
                }
        for(i=TT-1;i>0;i--)
        B[i]=B[i-1]^(V!=0)*(H[(F[G[i]]+F[V])%N]);
        B[0]=(V!=0)*(H[(F[G[0]]+F[V])%N]);
/*******************************************************************/
                        put(0,x);
                        put(1,x);

                CNT=(CNT+1)%N;
        }
        return (0) ;
}
```

APPENDIX C

RS Decoder Subroutine

```
/*******   Reed-Solomon Decoder   ***********************************/

#include <stdio.h>
#include "../type.h"
#include "../star.h"
#define TT       32
#define N        255
#define M        112
#define G        11
#define H        pstate->h
#define F        pstate->f
#define MUL(A, B)        ((B!=0)*(H[(A+F[(B)])%N]))
typedef struct {
        int non;
}
PARAM, *PARAMPTR;
typedef struct {
        unsigned char h[N],f[N+1];
}
STATE, *STATEPTR;
rsdec (pparam,size,pstate,pstar)
PARAMPTR pparam;
STATEPTR pstate;
STARPTR pstar;
int size;
{
        SAMPLE x;
        unsigned char et[17],ex[N],e[N],E[N+TT+1],S[TT+1],degR,degS;
        unsigned char R[TT+1],mu[TT+1],lam[TT+1],REC[N],tem,fa,fb;
        unsigned char *PR,*PS,*PT,*Plam,*Pmu,a,b;
        int i,j,L,CL,TH,ix,jx,i1,i2,j1,j2;

        if (pstate == NULL) {
        pstate = (STATEPTR) alloc_state_var(1,sizeof(STATE));
        if (no_input_fifos( ) !=1 || no_output_fifos( ) !=1)
                        return(3);

/*** H[ ] and F[ ] compute the power and log in GF(256)  ********/

                H[0]=1;
                for(i=0;i<8;i++) H[i+1]=2*H[i];
                for(i=8;i<N;i++) H[i]=H[i-1]^H[i-6]^H[i-7]^H[i-8];

                for(j=1;j<N+1;j++)  {
                        for(i=0;i<N;i++)  {
                                if(H[i]==j) F[j]=i;

                        }
                }

                F[0]=0;

        }
```

```
if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);

        while(length_input_fifo(0)  >0  )
        {
if (length_output_fifo(0)==maxlength_output_fifo(0)) return(0);
/**********************************************************************/
for(j=0;j<N;j++)
{
get(0,&x);
REC[j]=(char)x;
e[j]=0;
}

PR=R;
PS=S;
Plam=lam;
Pmu=mu;
for(j=0;j<=TT;j++) { R[j]=0; S[j]=0; lam[j]=0; mu[j]=0; }
/************ Syndrome calculation **********************************/
for(i=0;i<N;i++)
{
ix=N-1-i;
for(j=0;j<TT;j++) S[j]=REC[ix]^MUL(G*(j+M),S[j]);
}
degS=TT;
while(*(degS+PS)==0 && degS>0) --degS;
if(degS>0)
{

 /****** Modified Euclid Algorithm **********************************/
for(j=0;j<TT;j++) E[j+1]= *(PS+TT-1-j);
*(PR+TT)=1;
*mu=1;
degR=TT;
degS=TT;
i=1;
TH=TT/2;

while(i<=TT)
{
while(*(degR+PR)==0 && degR>0) --degR;
while(*(degS+PS)==0 && degS>0) --degS;

L=degR-degS;
CL=L;
if(L<0) CL= -L;

if(degR<TH || degS<TH)
        {
        if(degR>=TH)    Plam=Pmu;
        break;
        }
else
```

```
        {
        if(degR<degS)
                {
                PT=PR;
                PR=PS;
                PS=PT;
                PT=Plam;
                Plam=Pmu;
                Pmu=PT;
                tem=degR;
                degR=degS;
                degS=tem;
                }
        if(*(PS+degS)==0)
        {
        degS--;
        if(degS<TH)
                {
                Plam=Pmu;
                break;
                }
        }
        else
        {
/* compute R lam ************************************************************/
a= *(PR+degR);
b= *(PS+degS);
fa=F[a];
fb=F[b];
        degR--;
for(j=0;j<=TT;j++)
{
tem=(j>=CL)*(*(PS+j-CL));
PT=PR+j;
*PT=(b!=0)*MUL(fb,*PT)^(a!=0)*MUL(fa,tem);
tem=(j>=CL)*(*(Pmu+j-CL));
PT=Plam+j;
*PT=(b!=0)*MUL(fb,*PT)^(a!=0)*MUL(fa,tem);
}
/***************************************************************************/

        if(degR<TH) break;
        }
        }
        i++;
}
/*********** Error locator polynomial ***********************************/
degR=TT;
while(*(degR+Plam)==0 && degR>0) --degR;

tem=N-F[*(Plam+degR)];
for(j=0;j<=degR;j++)
        {
        PT=Plam+j;
        *PT=MUL(tem,*PT);
```

```
        }
/***************************************************************************/
for(j=TT;j<N+TT;j++)
        {
        E[j+1]=0;
        for(i=0;i<degR;i++)
                {
                tem= *(Plam+degR-i-1);
                jx=j-i;
                if(tem!=0) E[j+1] ^= MUL(F[tem], E[jx]);
                }
        }
E[0]=E[N];

for(j=1;j<=TT;j++)
if(E[j] != E[j+N]) {printf("ln * TEST FAILED ***"); j=0; break;}
if(j!=0)
{

/****************** Inverse FFT  ********************************/
for(j2=0;j2<15;j2++)
        {
        jx=G*17*j2;
        for(i1=0;i1<17;i1++)
                {
                et[i1]=0;
                for(i2=0;i2<15;i2++)
                        {
                        i=(N+1-M+8*(15*i1+17*i2))%N;
                        et[i1] ^= MUL(jx*i2,E[i]);
                        }
                }

        for(j1=0;j1<17;j1++)
                {
                ix=G*15*j1;
                        j=(15*j1+17*j2)%N;
                        e[j]=0;
                        for(i1=0;i1<17;i1++)
                                {
                                e[j] ^= MUL(ix*i1,et[i1]);
                                }
                }
        }

        for(j=0;j<N;j++) REC[j] ^= e[j];
}
}
for(j=0;j<N;j++)
        {
        x=(SAMPLE)REC[j];
        put(0,x);
        }
}
return (0) ;
```

APPENDIX D

Example of Output

i	u	i	u	i	u	i	u	
0	0	1	0	2	0	3	0	(Codeword)
4	0	5	0	6	0	7	0	
8	0	9	0	10	0	11	0	
12	0	13	0	14	0	15	0	
16	0	17	0	18	0	19	0	
20	0	21	0	22	0	23	0	
24	0	25	0	26	0	27	0	
28	0	29	0	30	0	31	0	
32	0	33	0	34	0	35	0	
36	0	37	0	38	0	39	0	
40	0	41	0	42	0	43	0	
44	0	45	0	46	0	47	0	
48	0	49	0	50	0	51	0	
52	0	53	0	54	0	55	0	
56	0	57	0	58	0	59	0	
60	0	61	0	62	0	63	0	
64	0	65	0	66	0	67	0	
68	0	69	0	70	0	71	0	
72	0	73	0	74	0	75	0	
76	0	77	0	78	0	79	0	
80	0	81	0	82	0	83	0	
84	0	85	0	86	0	87	0	
88	0	89	0	90	0	91	0	
92	0	93	0	94	0	95	0	
96	0	97	0	98	0	99	0	
100	0	101	0	102	0	103	0	
104	0	105	0	106	0	107	0	
108	0	109	0	110	0	111	0	
112	0	113	0	114	0	115	0	
116	0	117	0	118	0	119	0	
120	0	121	0	122	0	123	0	
124	0	125	0	126	0	127	0	
128	0	129	0	130	0	131	0	
132	0	133	0	134	0	135	0	
136	0	137	0	138	0	139	0	
140	0	141	0	142	0	143	0	
144	0	145	0	146	0	147	0	
148	0	149	0	150	0	151	0	
152	0	153	0	154	0	155	0	
156	0	157	0	158	0	159	0	
160	0	161	0	162	0	163	0	
164	0	165	0	166	0	167	0	
168	0	169	0	170	0	171	0	
172	0	173	0	174	0	175	0	
176	0	177	0	178	0	179	0	
180	0	181	0	182	0	183	0	
184	0	185	0	186	0	187	0	
188	0	189	0	190	0	191	0	
192	0	193	0	194	0	195	0	
196	0	197	0	198	0	199	0	

200	0	201	0	202	0	203	0
204	0	205	0	206	0	207	0
208	0	209	0	210	0	211	0
212	0	213	0	214	0	215	0
216	0	217	0	218	0	219	0
220	0	221	0	222	255	223	53
224	204	225	91	226	198	227	46
228	110	229	212	230	226	231	42
232	99	233	17	234	70	235	91
236	194	237	11	238	36	239	11
240	194	241	91	242	70	243	17
244	99	245	42	246	226	247	212
248	110	249	46	250	198	251	91
252	204	253	53	254	255		

No. of input errors = 2

i	s	log()	
0	16	4	(Syndrome)
1	117	136	
2	121	184	
3	191	134	
4	113	24	
5	55	53	
6	187	74	
7	243	96	
8	248	246	
9	218	239	
10	100	223	
11	66	88	
12	233	83	
13	30	43	
14	170	85	
15	151	145	
16	58	33	
17	190	22	
18	238	248	
19	33	87	
20	130	156	
21	254	60	
22	209	171	
23	133	159	
24	165	30	
25	188	182	
26	21	212	
27	107	63	
28	241	75	
29	74	173	
30	37	172	
31	16	4	

```
i=  1
d(R)= 32 d(S)= 31 L=  1
a=  0 b=  4
```

i	s	log()	r	log()		log()	u	log()
0	16	4	0	*	0	*	1	0
1	117	136	16	4	1	0	0	*
2	121	184	117	136	0	*	0	*
3	191	134	121	184	0	*	0	*
4	113	24	191	134	0	*	0	*
5	55	53	113	24	0	*	0	*
6	187	74	55	53	0	*	0	*
7	243	96	187	74	0	*	0	*
8	248	246	243	96	0	*	0	*
9	218	239	248	246	0	*	0	*
10	100	223	218	239	0	*	0	*
11	66	88	100	223	0	*	0	*
12	233	83	66	88	0	*	0	*
13	30	43	233	83	0	*	0	*
14	170	85	30	43	0	*	0	*
15	151	145	170	85	0	*	0	*
16	58	33	151	145	0	*	0	*
17	190	22	58	33	0	*	0	*
18	238	248	190	22	0	*	0	*
19	33	87	238	248	0	*	0	*
20	130	156	33	87	0	*	0	*
21	254	60	130	156	0	*	0	*
22	209	171	254	60	0	*	0	*
23	133	159	209	171	0	*	0	*
24	165	30	133	159	0	*	0	*
25	188	182	165	30	0	*	0	*
26	21	212	188	182	0	*	0	*
27	107	63	21	212	0	*	0	*
28	241	75	107	63	0	*	0	*
29	74	173	241	75	0	*	0	*
30	37	172	74	173	0	*	0	*
31	16	4	37	172	0	*	0	*
32	0	*	0	*	0	*	0	*

i= 2
d(R)= 31 d(S)= 31 L= 0
a=172 b= 4

0	16	4	217	176	37	172	1	0
1	117	136	176	192	16	4	0	*
2	121	184	199	204	0	*	0	*
3	191	134	103	47	0	*	0	*
4	113	24	240	46	0	*	0	*
5	55	53	156	133	0	*	0	*
6	187	74	134	27	0	*	0	*
7	243	96	46	226	0	*	0	*
8	248	246	251	23	0	*	0	*
9	218	239	52	128	0	*	0	*
10	100	223	212	179	0	*	0	*
11	66	88	124	245	0	*	0	*
12	233	83	180	150	0	*	0	*
13	30	43	137	9	0	*	0	*

14	170	85	99	186	0	*	0	*
15	151	145	114	70	0	*	0	*
16	58	33	83	138	0	*	0	*
17	190	22	231	193	0	*	0	*
18	238	248	185	236	0	*	0	*
19	33	87	65	155	0	*	0	*
20	130	156	7	106	0	*	0	*
21	254	60	174	37	0	*	0	*
22	209	171	148	174	0	*	0	*
23	1[illegible]3	159	202	77	0	*	0	*
24	1[illegible]5	30	173	11	0	*	0	*
25	188	182	119	247	0	*	0	*
26	21	212	11	188	0	*	0	*
27	107	63	72	208	0	*	0	*
28	241	75	219	38	0	*	0	*
29	74	173	169	146	0	*	0	*
30	37	172	177	115	0	*	0	*
31	16	4	0	*	0	*	0	*
32	0	*	0	*	0	*	0	*

i= 3
d(R)= 30 d(S)= 31 L= -1
a= 4 b=115

0	179	119	217	176	37	172	177	115
1	196	41	176	192	16	4	217	176
2	159	235	199	204	0	*	135	8
3	19	78	103	47	0	*	0	*
4	202	77	240	46	0	*	0	*
5	125	164	156	133	0	*	0	*
6	252	58	134	27	0	*	0	*
7	4	2	46	226	0	*	0	*
8	110	54	251	23	0	*	0	*
9	133	159	52	128	0	*	0	*
10	167	210	212	179	0	*	0	*
11	95	21	124	245	0	*	0	*
12	94	181	180	150	0	*	0	*
13	98	40	137	9	0	*	0	*
14	41	217	99	186	0	*	0	*
15	12	101	114	70	0	*	0	*
16	150	169	83	138	0	*	0	*
17	200	224	231	193	0	*	0	*
18	221	12	185	236	0	*	0	*
19	99	186	65	155	0	*	0	*
20	234	137	7	106	0	*	0	*
21	223	68	174	37	0	*	0	*
22	9	205	148	174	0	*	0	*
23	28	108	202	77	0	*	0	*
24	15	42	173	11	0	*	0	*
25	251	23	119	247	0	*	0	*
26	164	219	11	188	0	*	0	*
27	218	23[illegible]	72	208	0	*	0	*
28	57	69	219	38	0	*	0	*
29	53	177	169	146	0	*	0	*
30	169	146	177	115	0	*	0	*

31	0	*	0	*	0	*	0	*
32	0	*	0	*	0	*	0	*

i= 4
d(R)= 30 d(S)= 30 L= 0
a=146 b=115

0	32	5	217	176	37	172	2	1
1	107	63	176	192	16	4	227	154
2	0	*	199	204	0	*	147	123
3	0	*	103	47	0	*	0	*
4	0	*	240	46	0	*	0	*
5	0	*	156	133	0	*	0	*
6	0	*	134	27	0	*	0	*
7	0	*	46	226	0	*	0	*
8	0	*	251	23	0	*	0	*
9	0	*	52	128	0	*	0	*
10	0	*	212	179	0	*	0	*
11	0	*	124	245	0	*	0	*
12	0	*	180	150	0	*	0	*
13	0	*	137	9	0	*	0	*
14	0	*	99	186	0	*	0	*
15	0	*	114	70	0	*	0	*
16	0	*	83	138	0	*	0	*
17	0	*	231	193	0	*	0	*
18	0	*	185	236	0	*	0	*
19	0	*	65	155	0	*	0	*
20	0	*	7	106	0	*	0	*
21	0	*	174	37	0	*	0	*
22	0	*	148	174	0	*	0	*
23	0	*	202	77	0	*	0	*
24	0	*	173	11	0	*	0	*
25	0	*	119	247	0	*	0	*
26	0	*	11	188	0	*	0	*
27	0	*	72	208	0	*	0	*
28	0	*	219	38	0	*	0	*
29	0	*	169	146	0	*	0	*
30	0	*	177	115	0	*	0	*
31	0	*	0	*	0	*	0	*
32	0	*	0	*	0	*	0	*

i= 5
d(R)= 1 d(S)= 30 L=-29

i	o	log()	
0	37	172	(Error-locator polynomial)
1	16	4	
2	0	*	
3	0	*	
4	0	*	
5	0	*	
6	0	*	
7	0	*	
8	0	*	

9	0	*
10	0	*
11	0	*
12	0	*
13	0	*
14	0	*
15	0	*
16	0	*
17	0	*
18	0	*
19	0	*
20	0	*
21	0	*
22	0	*
23	0	*
24	0	*
25	0	*
26	0	*
27	0	*
28	0	*
29	0	*
30	0	*
31	0	*
32	0	*

i	e	log	i	e	log	i	e	log	i	e	log	
0	0	*	1	0	*	2	0	*	3	0	*	(Error
4	0	*	5	0	*	6	0	*	7	80	202	pattern)
8	0	*	9	0	*	10	0	*	11	0	*	
12	0	*	13	0	*	14	0	*	15	0	*	
16	0	*	17	0	*	18	0	*	19	0	*	
20	0	*	21	0	*	22	0	*	23	0	*	
24	0	*	25	0	*	26	0	*	27	0	*	
28	0	*	29	0	*	30	0	*	31	0	*	
32	0	*	33	0	*	34	0	*	35	0	*	
36	0	*	37	0	*	38	0	*	39	0	*	
40	0	*	41	0	*	42	0	*	43	0	*	
44	0	*	45	0	*	46	0	*	47	0	*	
48	0	*	49	0	*	50	0	*	51	0	*	
52	0	*	53	0	*	54	0	*	55	0	*	
56	0	*	57	0	*	58	0	*	59	0	*	
60	0	*	61	0	*	62	0	*	63	0	*	
64	0	*	65	0	*	66	0	*	67	0	*	
68	0	*	69	0	*	70	0	*	71	0	*	
72	0	*	73	0	*	74	0	*	75	0	*	
76	0	*	77	0	*	78	0	*	79	0	*	
80	0	*	81	0	*	82	0	*	83	0	*	
84	0	*	85	0	*	86	0	*	87	0	*	
88	0	*	89	0	*	90	0	*	91	0	*	
92	0	*	93	0	*	94	0	*	95	0	*	
96	0	*	97	0	*	98	0	*	99	0	*	
100	0	*	101	0	*	102	0	*	103	0	*	
104	0	*	105	0	*	106	0	*	107	0	*	
108	0	*	109	0	*	110	0	*	111	0	*	

112	0	*	113	0	*	114	0	*	115	0	*
116	0	*	117	0	*	118	0	*	119	0	*
120	1	0	121	0	*	122	0	*	123	0	*
124	0	*	125	0	*	126	0	*	127	0	*
128	0	*	129	0	*	130	0	*	131	0	*
132	0	*	133	0	*	134	0	*	135	0	*
136	0	*	137	0	*	138	0	*	139	0	*
140	0	*	141	0	*	142	0	*	143	0	*
144	0	*	145	0	*	146	0	*	147	0	*
148	0	*	149	0	*	150	0	*	151	0	*
152	0	*	153	0	*	154	0	*	155	0	*
156	0	*	157	0	*	158	0	*	159	0	*
160	0	*	161	0	*	162	0	*	163	0	*
164	0	*	165	0	*	166	0	*	167	0	*
168	0	*	169	0	*	170	0	*	171	0	*
172	0	*	173	0	*	174	0	*	175	0	*
176	0	*	177	0	*	178	0	*	179	0	*
180	0	*	181	0	*	182	0	*	183	0	*
184	0	*	185	0	*	186	0	*	187	0	*
188	0	*	189	0	*	190	0	*	191	0	*
192	0	*	193	0	*	194	0	*	195	0	*
196	0	*	197	0	*	198	0	*	199	0	*
200	0	*	201	0	*	202	0	*	203	0	*
204	0	*	205	0	*	206	0	*	207	0	*
208	0	*	209	0	*	210	0	*	211	0	*
212	0	*	213	0	*	214	0	*	215	0	*
216	0	*	217	0	*	218	0	*	219	0	*
220	0	*	221	0	*	222	0	*	223	0	*
224	0	*	225	0	*	226	0	*	227	0	*
228	0	*	229	0	*	230	0	*	231	0	*
232	0	*	233	0	*	234	0	*	235	0	*
236	0	*	237	0	*	238	0	*	239	0	*
240	0	*	241	0	*	242	0	*	243	0	*
244	0	*	245	0	*	246	0	*	247	0	*
248	0	*	249	0	*	250	0	*	251	0	*
252	0	*	253	0	*	254	0	*			

BLOCK= 0 SYMERR= 0 BITERR= 0

APPENDIX E

Example of Output for Randomly Chosen Codeword

i	u	i	u	i	u	i	u	
0	2	1	143	2	104	3	10	(Codeword)
4	157	5	95	6	32	7	68	
8	139	9	200	10	143	11	186	
12	130	13	130	14	240	15	26	
16	124	17	179	18	133	19	176	
20	222	21	164	22	4	23	211	
24	42	25	248	26	131	27	219	
28	37	29	92	30	227	31	151	
32	135	33	187	34	189	35	61	
36	36	37	246	38	205	39	227	
40	155	41	19	42	223	43	111	
44	130	45	245	46	58	47	12	
48	13	49	141	50	143	51	31	
52	65	53	144	54	96	55	187	
56	154	57	172	58	239	59	148	
60	59	61	55	62	172	63	113	
64	154	65	106	66	26	67	38	
68	167	69	14	70	69	71	3	
72	128	73	182	74	165	75	201	
76	148	77	111	78	142	79	30	
80	88	81	246	82	29	83	130	
84	107	85	106	86	162	87	197	
88	72	89	133	90	141	91	238	
92	111	93	71	94	216	95	201	
96	17	97	52	98	246	99	192	
100	62	101	1	102	62	103	95	
104	141	105	203	106	215	107	66	
108	79	109	203	110	32	111	138	
112	49	113	136	114	165	115	209	
116	115	117	141	118	129	119	162	
120	139	121	157	122	81	123	100	
124	86	125	101	126	158	127	215	
128	195	129	180	130	199	131	5	
132	253	133	41	134	111	135	177	
136	27	137	109	138	107	139	84	
140	72	141	226	142	39	143	138	
144	112	145	220	146	156	147	9	
148	111	149	81	150	177	151	20	
152	165	153	14	154	50	155	112	
156	135	157	108	158	52	159	216	
160	133	161	132	162	2	163	237	
164	252	165	224	166	142	167	178	
168	126	169	180	170	87	171	121	
172	21	173	143	174	217	175	88	
176	234	177	142	178	120	179	33	
180	117	181	17	182	235	183	211	
184	39	185	243	186	39	187	140	
188	151	189	54	190	207	191	3	
192	45	193	61	194	30	195	116	
196	79	197	128	198	79	199	29	
200	13	201	189	202	145	203	43	

204	77	205	172	206	108	207	47
208	118	209	54	210	177	211	22
212	155	213	40	214	226	215	153
216	44	217	101	218	37	219	51
220	28	221	156	222	167	223	175
224	80	225	108	226	20	227	122
228	202	229	251	230	31	231	81
232	29	233	89	234	159	235	134
236	60	237	217	238	91	239	13
240	243	241	103	242	83	243	142
244	162	245	69	246	174	247	177
248	55	249	20	250	163	251	108
252	191	253	74	254	141		

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. JPL Pub. 85-23	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle A Software Simulation Study of a (255,223) Reed-Solomon Encoder/Decoder		5. Report Date April 15, 1985
		6. Performing Organization Code
7. Author(s) Fabrizio Pollara		8. Performing Organization Report No.
9. Performing Organization Name and Address JET PROPULSION LABORATORY California Institute of Technology 4800 Oak Grove Drive Pasadena, California 91109		10. Work Unit No.
		11. Contract or Grant No. NAS7-918
		13. Type of Report and Period Covered JPL Publication
12. Sponsoring Agency Name and Address NATIONAL AERONAUTICS AND SPACE ADMINISTRATION Washington, D.C. 20546		14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

A set of software programs which simulates a (255,223) Reed-Solomon encoder/decoder pair is described. The transform decoder algorithm uses a modified Euclid algorithm, and closely follows the pipeline architecture proposed for the hardware decoder. Uncorrectable error patterns are detected by a simple test, and the inverse transform is computed by a finite field FFT.

Numerical examples of the decoder operation are given for some test codewords, with and without errors. The use of the software package is briefly described.

17. Key Words (Selected by Author(s))	18. Distribution Statement
Communications Information Theory Voyager Project	Unlimited/unclassified

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified		

JPL 0184 R 9/83